AFRL-RI-RS-TR-2014-268

CLOUD LIBRARY FOR DIRECTED PROBABILISTIC GRAPHICAL MODELS

THE BOEING COMPANY

OCTOBER 2014

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 88th ABW, Wright-Patterson AFB Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2014-268 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/ S /
NANCY ROBERTS
Work Unit Manager

/ S /
MICHAEL J. WESSING
Deputy Chief, Information Intelligence
Systems and Analysis Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
OCTOBER 2014	FINAL TECHNICAL REPORT	SEP 2012 – DEC 2013

4. TITLE AND SUBTITLE

CLOUD LIBRARY FOR DIRECTED PROBABILISTIC GRAPHICAL MODELS

5a. CONTRACT NUMBER
FA8750-12-C-0332

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
62702F

6. AUTHOR(S)

Haiqin Wang and Marek Druzdzel

5d. PROJECT NUMBER
XDAT

5e. TASK NUMBER
A0

5f. WORK UNIT NUMBER
04

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

The Boeing Company
PO Box 3707, M/S 4C-76
Seattle, WA 98124-2207

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RIEA
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2014-268

12. DISTRIBUTION AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited. PA# 88ABW-2014-4715
Date Cleared: 8 OCT 2014

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The project aimed to build a massively parallel library for Bayesian networks by creating a data analytical capability with potential throughput commensurate with DoD data volumes. The goal was to implement data-parallel independent & identically distributed inference & learning in Bayesian networks & accomplish nearly-linear scaling. They reexamined & implemented data structures & algorithms needed for distributed-model inference. The inference aimed at being able to ask & answer privacy & adversarial learning questions where model distribution is due to private nature of the data. They looked for efficiently-parallelizable methods of inference & learning.

15. SUBJECT TERMS
Bayesian Networks, cloud computing, probabilistic graphical models, cloud analytics

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**NANCY ROBERTS**
U	U	U	UU	16	19b. TELEPHONE NUMBER (*Include area code*) **315-330-3566**

Table of Contents

List of Figures

1. Summary

Bayesian networks are a mature statistical model with a number of traditional implementations. However, there are no open-source implementations of graphical model inference and learning that take advantage of distributed hardware in a transparent fashion. Such graphical model inference and learning capabilities are currently being built ad-hoc to suit a particular application-hardware combination.

The project aimed at building a massively parallel library for Bayesian networks to create a data analytical capability with potential throughput commensurate with DoD data volumes. Our goal was to implement data-parallel independent and identically distributed (i.i.d.) inference and learning in Bayesian networks and accomplish nearly-linear scaling. We proposed to reexamine and efficiently implement data structures and algorithms needed for distributed-model inference. The inference aimed at being able to ask and answer privacy and adversarial learning questions where model distribution is due to private nature of the data. We looked for efficiently-parallelizable methods of inference and learning, where graphical models are defined over arbitrary large graphs such as social networks but parameters are possibly tied across long distances in the graph.

2. Introduction

There are three classes of challenges in bringing Bayesian network algorithms into the massively parallel world, each of which necessitates a different approach.

1. Instance-parallelizable algorithms, where multiple working instances of existing algorithms process independent instance of data in parallel, come with the lowest technical risk and are addressed by a more or less direct application of MapReduce and related paradigms.
2. The case where a problem instance with data cannot, for technical or privacy reasons, be brought into memory in its entirety. This case requires new research to create robust versions of well-known fundamental graphical model data structures that remain cohesive when spanning over multiple computing nodes. The algorithms and data structures are not new in principle, but they include hidden assumptions that are harmless in the single-threaded case. Extending the representations and algorithms to support parallelism requires a thorough examination of the algorithms and the data structures.
3. Dealing with social graph-based information, we face a relational learning paradigm where data is not organized into instances, but instead statistical commonalities arise between areas of the graph. This is a developing area and both conceptual research and careful implementation are needed.

Our approach was to build on top of and integrate existing technology where it exists and is appropriate, for example, Spark for iterative algorithms, GraphLab for graph-structured computations.

The library aimed to be integrated into popular big-data analytical software like Hive to lower developer friction, leveraging both the data access primitives in Hive and BayesDB, and the interaction languages.

3. Methods, Assumptions, and Procedures

3.1 SMILE-WIDE v1.0

While the long term goal has included design and implementation of a new, open source, Java-based library for inference and learning in Bayesian networks (our steps to this effect are described later in the report), we started the project based on SMILE, the existing implementation of the software developed by the Decision Systems Laboratory, University of Pittsburgh. We have utilized the jSMILE library as the foundation for parallel inference with i.i.d. data in order to establish performance baselines. This way, we were able to perform parallelization experiments while pursuing the design and implementation efforts for the ultimate goal.

We called the library based on jSMILE, SMILE-WIDE 1.0. We tested the code with Boeing's mini-cluster and then transferred to the XNET integration environment. We established a build process for the library .jars on XNET and exposed the library API as a standard .jar library. The API is similar to the jSMILE interface, but it is extended so that the potentially long-running calls are being translated into Hadoop jobs. The deployment and functionality test of the initial version of the library on XNET was successful, including the native SMILE binary.

3.2 Performance Test with Facebook Dataset

We conducted scale-up testing of the parallel inference code on the XNET cluster using the Facebook Users dataset. The task for this data set was to predict age group from other attributes of the Facebook sub dataset. We tested the prototype for parallel inference speedup achievable on the XNET cluster. It parsed the Facebook dataset and distributed computation to all available nodes. From small-scale testing on Boeings mini-cluster it appeared that the speedups were nearly linear (optimal), as expected for i.i.d. inference. In the test on the XNET cluster, there were 37 16-core nodes, 592 CPUs, 446 mapper compute threads. Again, the speedups were nearly linear until the jobs setup overhead dominated runtime.

The result is shown in the chart below.

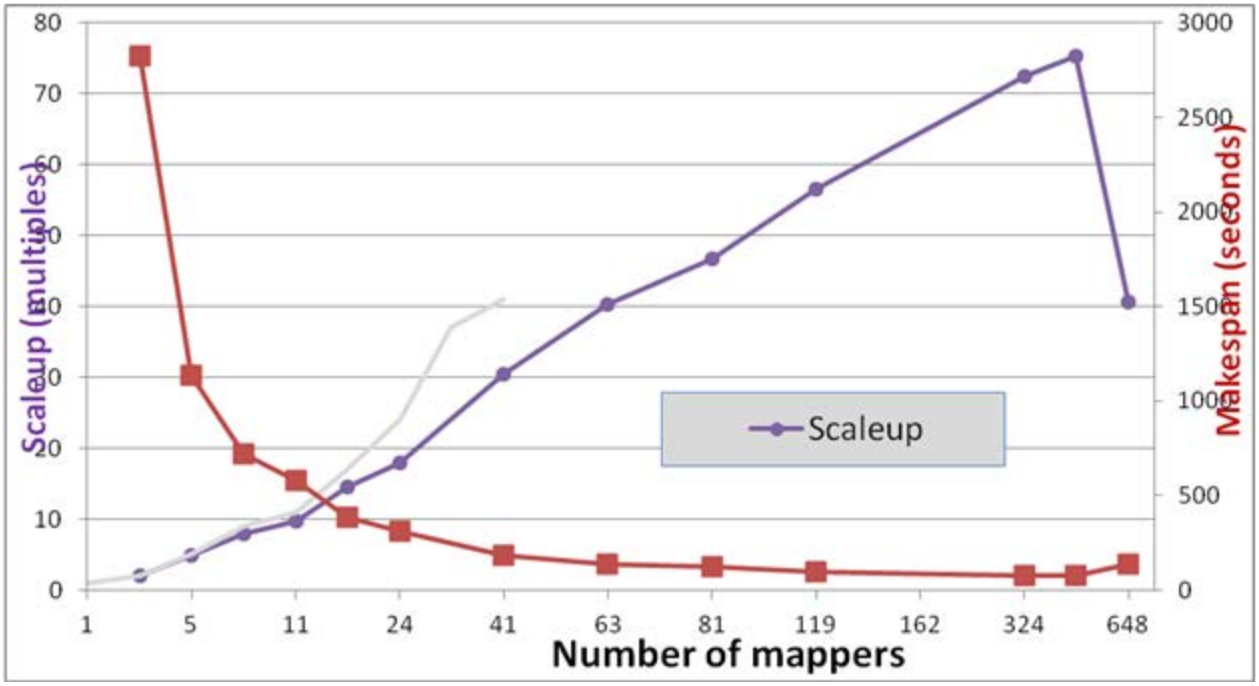

Figure 1: Runtimes and Speedups for Parallel Bayesian Network Inference

Generally, the runtime reduces and the speedup increases when the number of mappers increases for parallel Bayesian network inference. For example, when there are 17 mapper threads, the scale-up slope shows 15 times of increase rate compared with single mapper thread. However, when the number of mappers reaches 432, the job setup overhead, which is about 45 seconds, dominates runtime. The performance decreases when the number of mappers is greater than 432 as the Mappers start to queue and there is a delay in processing the queued jobs. For example, when there are 648 mapper threads, the scale-up slope showed only 41 times increase of speedups for parallel Bayesian network inference.

3.3 Hive UDFs

Originally defining an inference process on top of a dataset means writing/changing ~200 lines of code against our library. Much of this is unavoidable as it is the input dataset parser, but some could be generalized and simplified, especially data munging. We explored, with other XDATA API groups, common ways of processing data across XDATA framework such as normalizing text inputs, discretizing continuous attributes, etc. The data access drives most of the programming complexity/effort for the app developer against our library. As a result, we incorporated data access models by exposing the functionality as a set of Hive UDFs (user-defined functions), which allows tight integration of the library with Hive data warehouse for other jobs such as Hive data mining.

3.4 Structure Learning and the EM algorithm

Continuing to develop the initial version of the library, we have added the structure learning components, as well as parallel cross-validation (evaluation routine). We also used the XNET cluster to research the distribution of Bayesian network structure scores and the connection between the scores and actual predictive performance. We redesigned the Expectation-Maximization (EM) algorithm so that it relies in its crucial step on a general-purpose "counting class" – the collection of expected counts of arbitrary subsets of attributes. These correspond to Bayesian network "families." The EM algorithm allows us to learn from incomplete data at scale and to implement algorithms like probabilistic clustering.

We performed experiments with the parallelized version of the EM algorithm on the XDATA cluster on a data set containing 500,000 records. We were interested in the relationship between the number of Mappers and the time required by the EM algorithm to converge. The plot below shows this relationship,

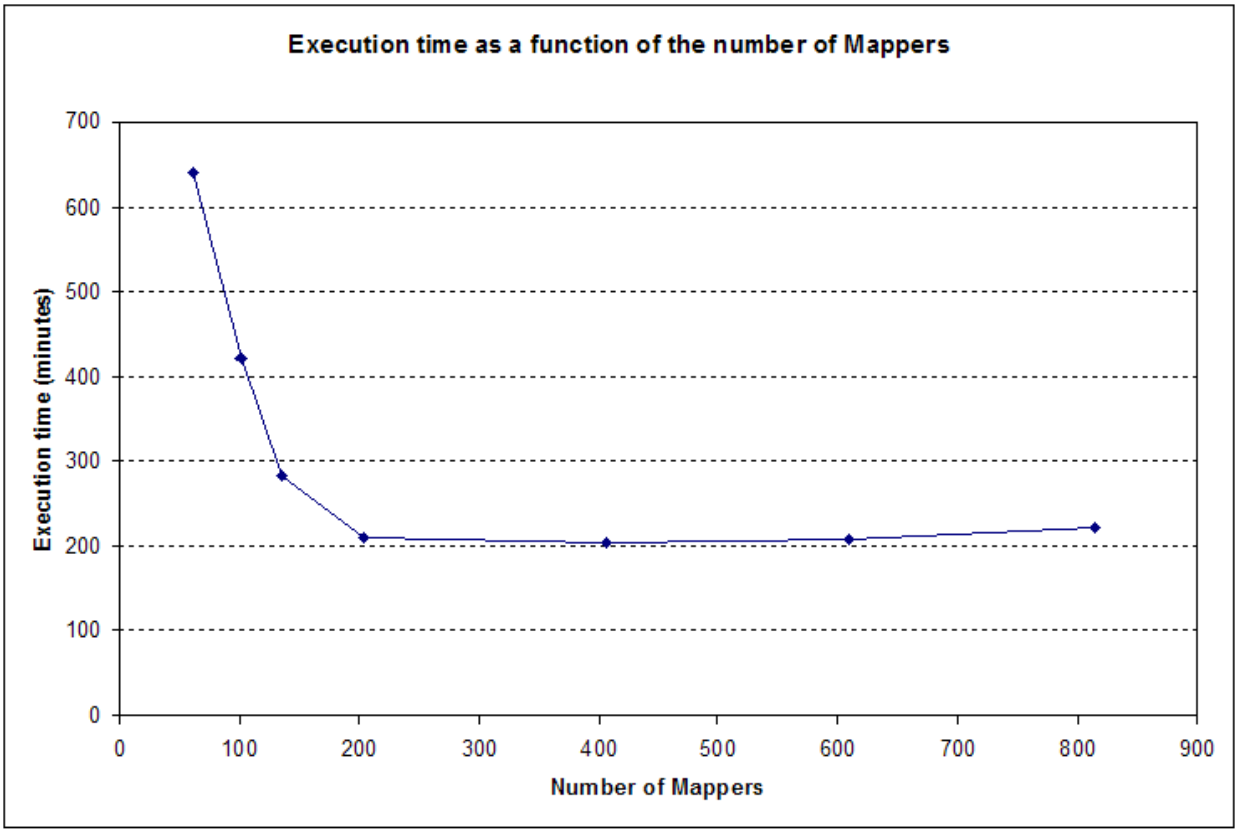

Figure 2: Execution time as a function of the number of Mappers

We can observe that there is a dramatic speedup due to parallel execution of the EM algorithm but only up to a certain point. We believe that the overhead related to setting up and coordinating the Map-Reduce jobs and, in particular data transfer between Mappers and Reducers, outweighs the computational gains. At the time of our experiments, the XDATA

cluster could allocate up to 444 Mappers. This is a critical point, as our algorithm spends most of its time in the Map phase, which generates sufficient statistics. The number of Reducers requested by the algorithm is equal to the number of nodes, which in our network was less than 20. The key to achieving effective speedup was the use of the Combiner class. The Combiner class is used by the Map-Reduce framework to perform local aggregation of the intermediate outputs, which helps to cut down the amount of data transferred from the Mappers to the Reducers.

A next step would be to conduct experiments using Spark.

3.5 Analysis of the Kiva Dataset

Our practical data analytic work was mostly with the Kiva dataset. The Bitcoin and Akamai data sets are deeply graph-structured and are thus not a natural fit for what is essentially a probabilistic model making i.i.d. assumptions and would require heavy feature engineering.

We parsed the Kiva data set and mapped its values onto outcomes of random variables. Then we deployed Bayesian structure search algorithms to find a Bayesian network that best describes the dataset. During the summer workshop, we applied SMILE-WIDE to the Kiva dataset, exercising Bayesian network structure learning algorithms and probabilistic clustering through learning a latent variable graphical model, using the distributed EM capability.

The structures learned from the data validated common-sense, notably that geographical variables are closely related, partners operate in distinct countries, currency correlates with country, delinquency is strongly country-dependent, loan amounts (terms/funded/paid) are closely correlated, and amounts co-vary with sector.

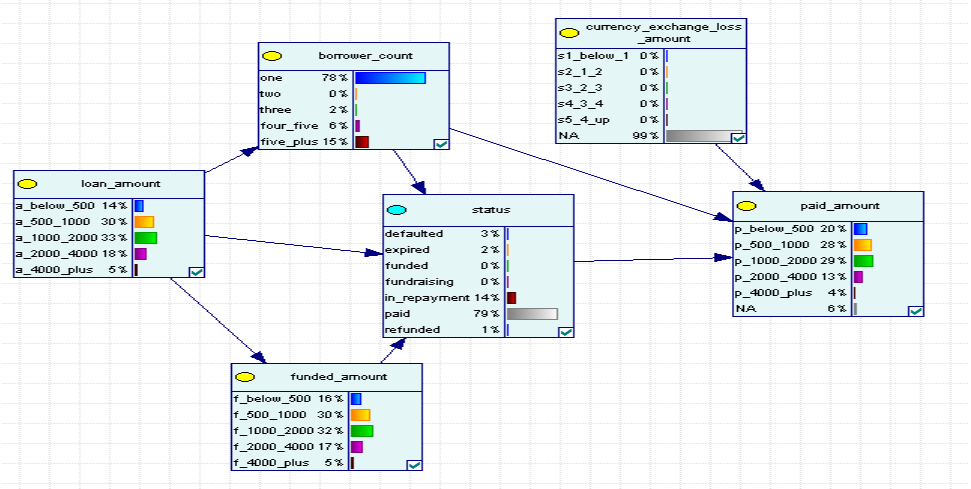

Figure 3: Learned Bayesian Network

The learned Bayesian network structure (above) allows for answering many queries of interest to analysts. We have found when asking various queries that when the number of borrowers is between 3 and 5, they are almost twice as likely to default as when the number of borrowers is 1 or 2. We have found that loans with the number of borrowers larger than 5 rarely default. Defaulting loans are likely to involve small amounts of money.

3.6 Algorithms Research

In parallel with exploring instance-parallelizable algorithms, we started research on truly parallel learning algorithms in order to make contributions to parallelizing both the Bayesian search and the constraint-based search approach. Some avenues for this work were through the idea of random forests and Random Naive Bayes ensemble learning. Our goal was to create a MapReduce Bayesian network learning ensemble algorithm for classification purposes. We intended to apply bootstrapping and random feature selection to construct a forest of classifiers with varying network complexity ranging from Naive Bayes, Tree Augmented Naive Bayes, Augmented Bayes, to full blown Bayesian network learning without restrictions on network structure. Part of this work was focused on determining the optimal way of combining the output of the individual classifiers (Bayesian Model Averaging vs. Bayesian Model Combining vs. possible novel approaches).

We investigated the question whether a massive amount of computation is capable of improving the results of learning algorithms. In the Bayesian Search approach, instance-based parallelism is fairly straightforward to achieve by distributing algorithm restarts among Mappers. It turns out that with massively many restarts, the score of the identified structure increases and so does classification accuracy. The following two plots show histograms of structure scores (upper plot) and classification accuracy (lower plot). Massive amount of computation allows for finding structures that are outliers in the sense of being in the right-hand tail of the distribution and having scores that are significantly higher than the average.

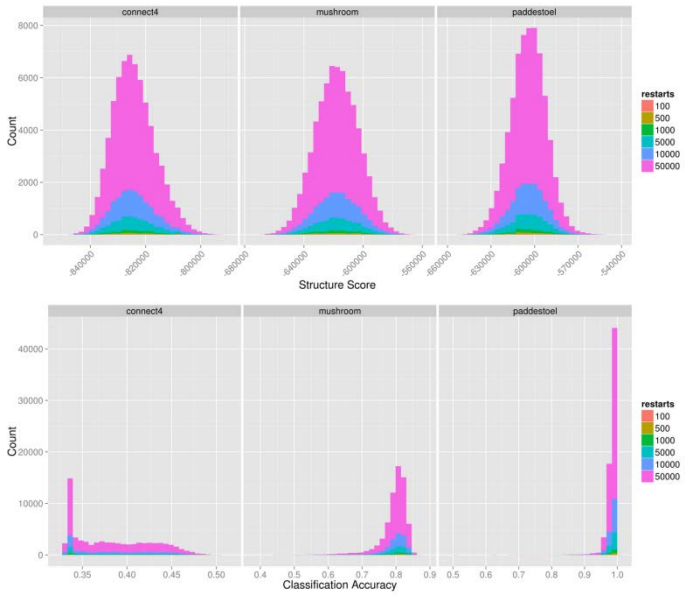

Figure 4: Histograms of structure scores (upper) and classification accuracy (lower)

We also investigated the relationship between structure score and classification accuracy (below).

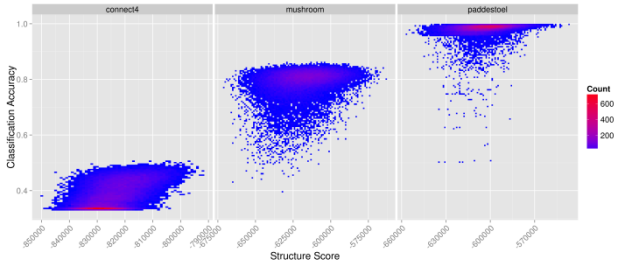

Figure 5: Relationship between structure score and classification accuracy

It turns out that while the relationship is not strong, higher scores, achieved by massive amounts of computation, are correlated with higher accuracy.

Finally, we studied an important parameter of the search algorithm, notably the average node degree, which is the sum of in-degree and out-degree or the total number of arcs through which the node is connected to the rest of the network. We plot below the relationships between node degree and structure score (upper plot) and classification accuracy (lower plot).

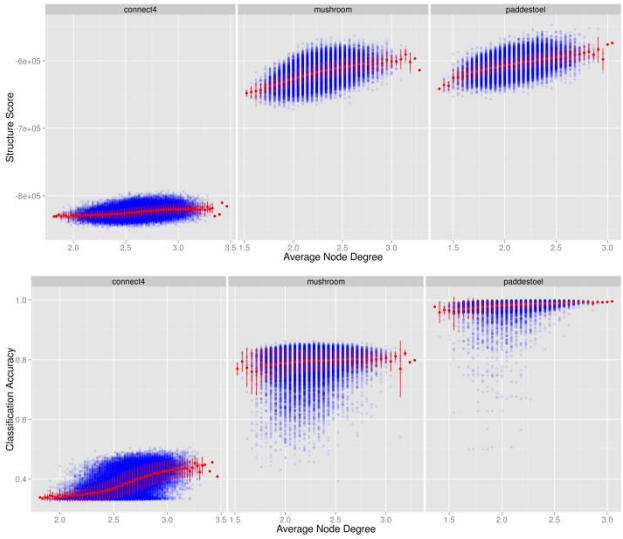

Figure 6: Relationships between node degree and structure score (upper) and classification accuracy (lower)

It is clear that both cases we are dealing with positive relationships, which justifies computational effort spent on massively parallel search.

The SMILE-WIDE 1.0 library includes four general Map-Reduce Bayesian network structure learning algorithms. They are: Two different MapReduce versions of the PC algorithm, a MapReduce version of the Bayesian Search algorithm, and a version of the Three-Phase Dependency Analysis algorithm.

A next step would be to improve algorithm efficiency. We were considering algorithm modifications that allow for reduction of the number of required MapReduce jobs, the number of messages that need to be communicated between Mappers and Reducers, and the memory usage of Mappers and Reducers. All these seem to be the current bottlenecks of the implemented algorithms. Assuming that the search performed on a single processor has sufficient memory available, it outperforms the Hadoop implementation.

For instance, the exponential nature of the number of required independence tests of the PC algorithm, potentially causes problems with the number of mapper messages (Key-Value Pairs, (KVPs)) sent by mapper process in our completely distributed independence test code.

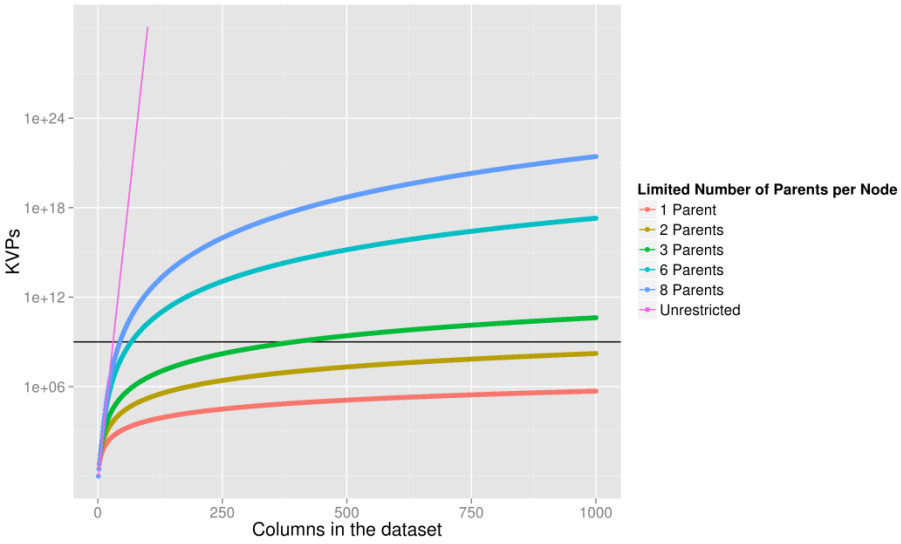

Figure 7: Key-Value Pairs vs Columns in the Dataset

The number of messages that will be passed by the mappers can be curbed by reducing the number parent nodes (for Bayesian search based algorithms) or the number of conditioning variables (for constraint-based search based approaches), but the number will still quickly become unacceptable to the point that Hadoop clusters do not have sufficient temporary storage to complete the map phase of the algorithm.

3.7 Results and Discussion

We have set out to design and develop SMILE-WIDE, an open source Java library for Bayesian network inference and learning as the main part of the project. At the end of October 2013, we had the design and a partial implementation of the library. We have implemented the structural part for representing discrete Bayesian networks and XML-based input-output, which is compatible with SMILE file format (.XDSL). The library is accompanied by unit tests. The project is a self-contained Maven project and all dependencies are resolved from public repositories.

Library documentation is created using Javadoc, which allows for automatic generation of manual pages. The source code and documents are cleaned up for generality and robustness. They are checked into Git repository. Recently project web site was moved to Github too.

Git repository: https://github.com/SmileWide/main
Github wiki site: http://smilewide.github.io/main

4. Conclusions / Recommendations

We implemented a basic version of SMILE-WIDE library and tested it for instance-parallelizable algorithms for both inference and learning of Bayesian networks. The test results showed the parallel algorithms can greatly speedup the execution time but only up to certain limit. The overhead of distribution of jobs between Mappers and Reducers, the communication load by the number of messages needed to exchange between Mappers and Reducers, and memory usage of Mappers and Reducers, seem to be the bottleneck. We believe computation efficiency can be further improved with these issues addressed for parallel algorithms and implementations.

5. List of Acronyms

i.i.d – independent and identically distributed

API – application programming interface

CPU – central processing unit

DARPA – Defense Advanced Research Projects Agency

DoD - Department of Defense

EM – Expectation Maximization

KVP – key value pairs

PC – Peter and Clark(authors of the algorithm)

SMILE - Structural Modeling, Inference, and Learning Engine

SMILE-WIDE - Structural Modeling, Inference, and Learning Engine - With Integrated Distributed Execution

UDF – user-defined functions

XML – eXtensible Markup Language